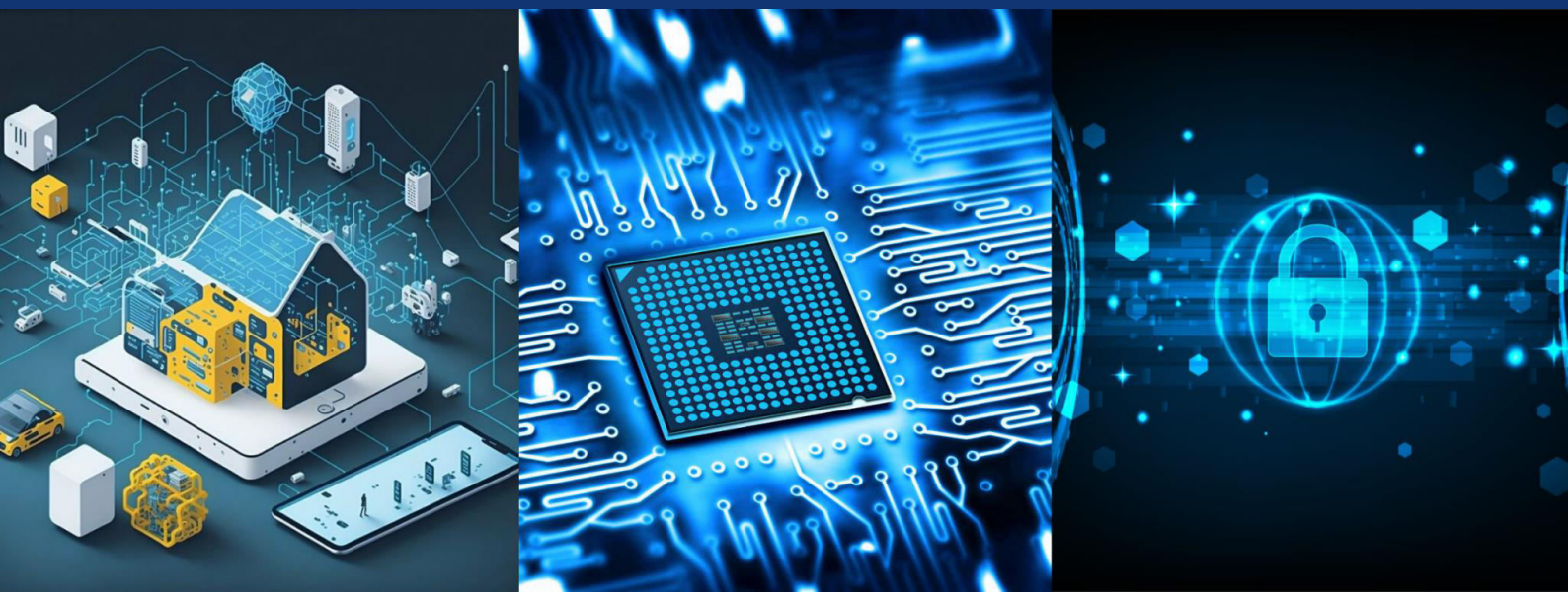
IJIRCCE

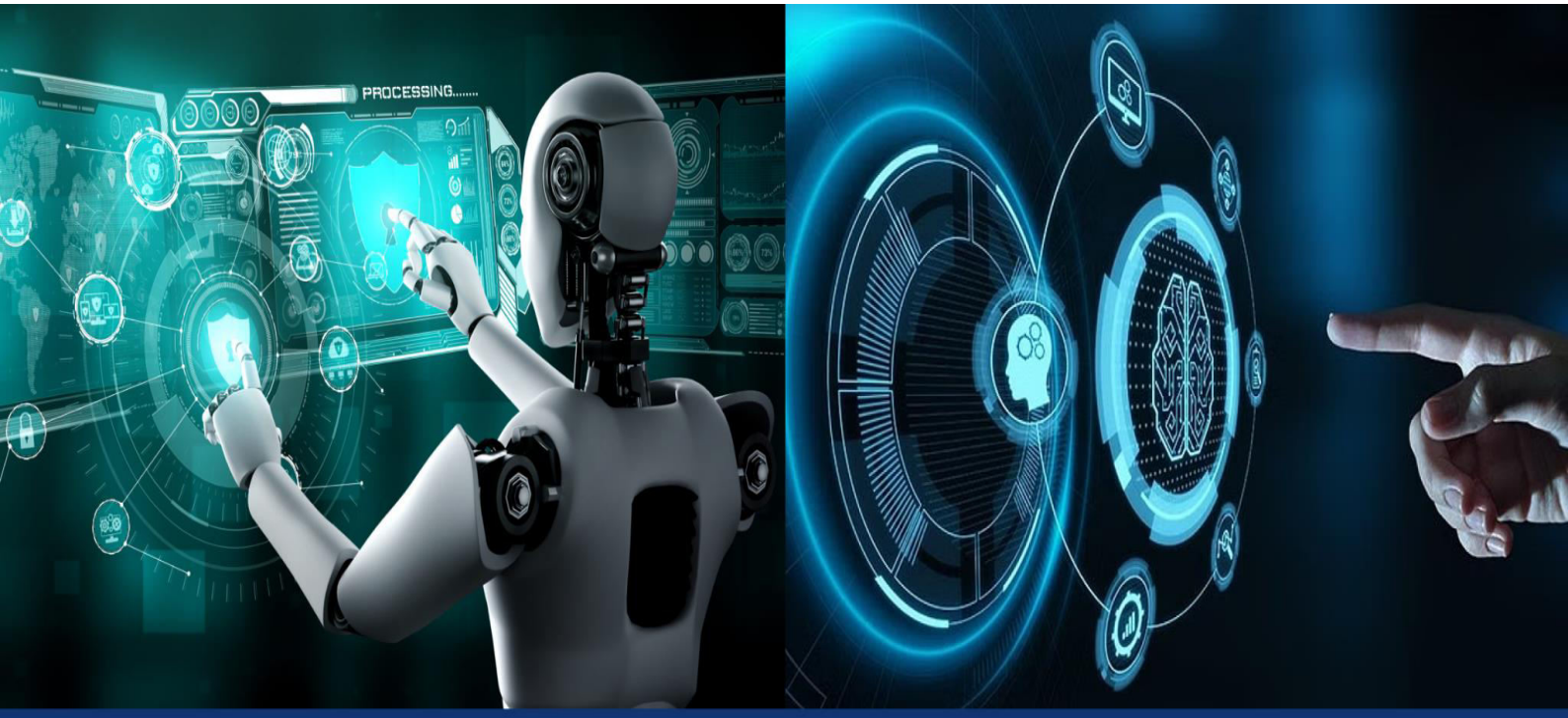
ISSN(O): 2320-9801

ISSN(P): 2320-9798



International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.927

Volume 14, Issue 7, July 2026

© 2026 IJIRCCE | Volume 14, Issue 7, July 2026 DOI: 10.15680/IJIRCCE.2026.1407057

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Hybrid Static–Dynamic Analysis Framework for Detecting SMS-OTP Interception Malware on Android Devices

Dr. R. Sridevi, Soumya Erra

Professor, Department of Computer Science Engineering, Jawaharlal Nehru Technological University, Hyderabad, Telangana, India

Post-Graduate Student, Department of Computer Science Engineering, Jawaharlal Nehru Technological University, Hyderabad, Telangana, India

ABSTRACT: The rapid growth of Android applications has significantly increased the risk of malware capable of intercepting SMS-based One-Time Passwords (OTPs), leading to financial fraud, identity theft, and unauthorized access to sensitive user accounts. Traditional Android malware detection approaches primarily rely on static permission analysis or signature-based detection techniques, which are often insufficient to identify emerging malware variants and applications requesting excessive sensitive permissions. To address these limitations, this paper proposes a Hybrid Static–Dynamic Analysis Framework for Detecting SMS-OTP Interception Malware on Android Devices. The proposed framework integrates Python, Flask, Androguard, Android Studio, Java, Android SDK, and Android PackageManager APIs to perform APK analysis, permission extraction, installed application analysis, risk score calculation, and security report generation. Static analysis examines uploaded APK files by extracting manifest information, application components, and sensitive permissions, while the device-level analysis module evaluates installed Android applications using PackageManager APIs to identify risky permissions and calculate security scores. The framework classifies applications into Safe, Medium Risk, and High Risk categories based on predefined risk assessment rules and generates comprehensive security reports for end users. Experimental results demonstrate that the proposed framework effectively identifies security-sensitive applications, improves Android security awareness, and provides an economical solution for Android malware assessment suitable for academic research and practical mobile security applications.

KEYWORDS: Android Malware Detection, SMS-OTP Interception, Static Analysis, Device-Level Security Assessment, Android Security, APK Analysis, Risk Assessment, Android PackageManager, Flask, Androguard.

I. INTRODUCTION

The increasing popularity of Android smartphones has made them one of the primary targets for cybercriminals. Millions of Android applications are installed daily, providing users with numerous services such as banking, online shopping, digital payments, communication, and authentication. However, many malicious applications exploit Android permissions to intercept SMS messages, steal One-Time Passwords (OTPs), monitor user activities, and gain unauthorized access to sensitive accounts.

SMS-based OTP authentication remains one of the most widely adopted security mechanisms for banking, financial transactions, e-commerce platforms, and online authentication services. Malware capable of reading, receiving, forwarding, or manipulating SMS messages can bypass this authentication mechanism and compromise user privacy and financial security. Recent Android malware families increasingly employ permission abuse, code obfuscation, overlay attacks, and accessibility service misuse to avoid detection while collecting sensitive user information.

Existing Android malware detection techniques mainly focus on static permission analysis or signature-based detection. Although these approaches successfully detect known malware, they often fail to provide comprehensive device-level security assessment and detailed risk analysis of installed applications. Furthermore, users receive limited information regarding the actual security risks associated with sensitive permissions requested by applications.

© 2026 IJIRCCE | Volume 14, Issue 7, July 2026 DOI: 10.15680/IJIRCCE.2026.1407057

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

This paper proposes a Hybrid Static–Dynamic Analysis Framework for Detecting SMS-OTP Interception Malware on Android Devices. The proposed framework integrates static APK analysis, device-level installed application analysis, permission extraction, risk score calculation, security classification, and report generation within a unified Android security assessment platform. The framework is implemented using Python, Flask, Androguard, Android Studio, Java, and Android SDK, providing users with an efficient, practical, and user-friendly solution for Android application security analysis.

II. RELATED WORK

The increasing use of mobile banking, digital payment systems, and online authentication has made Android devices one of the primary targets for cyberattacks. Among various threats, malware capable of intercepting SMS messages and One-Time Passwords (OTPs) has become a major security concern. To address this issue, Zhao et al. investigated implementation flaws in SMS-OTP authentication mechanisms and demonstrated that malicious applications with SMS access can intercept authentication messages and compromise user accounts. Their study highlights the importance of identifying applications requesting SMS-related permissions and protecting users against OTP interception attacks [1].

To provide a secure software ecosystem, Android incorporates multiple built-in security mechanisms including application sandboxing, permission-based access control, and application signing. Enck et al. presented one of the earliest comprehensive studies on Android security architecture, explaining how the permission model restricts application access to sensitive resources. Their work also emphasized that applications requesting unnecessary or excessive permissions could still introduce serious security vulnerabilities if users unknowingly grant those permissions [2]. Building upon the Android permission model, researchers introduced permission-based static analysis for Android malware detection. This technique examines the AndroidManifest.xml file to identify sensitive permissions such as READ_SMS, RECEIVE_SMS, SEND_SMS, READ_PHONE_STATE, and READ_CONTACTS, which are frequently associated with malicious applications. Since this approach analyses applications without executing them, it offers fast and computationally efficient malware detection. However, permission analysis alone cannot determine whether an application actually misuses these permissions during execution, which may lead to inaccurate classification results [2]. To improve the effectiveness of static analysis, Arp et al. proposed the DREBIN malware detection framework. DREBIN extracts multiple lightweight static features, including permissions, hardware components, API calls, application metadata, and network addresses, before applying machine learning algorithms for malware classification. The framework achieved high detection accuracy while maintaining low computational overhead, making it suitable for mobile environments. Furthermore, DREBIN provides explainable detection by identifying the features responsible for classifying an application as malicious, thereby improving transparency and analyst understanding [3]. Although feature-based static analysis significantly improves malware detection, identifying information leakage within Android applications requires deeper code analysis. To address this challenge, Arzt et al. introduced FlowDroid, a precise static taint analysis framework that traces sensitive data flows from information sources to potential leakage points. Unlike conventional permission analysis, FlowDroid considers Android lifecycle methods, callback functions, object sensitivity, and context sensitivity to accurately detect privacy leaks. While the framework provides detailed program analysis, its computational complexity makes it more suitable for offline security assessment than lightweight mobile deployment [4].

Beyond static analysis, researchers have investigated the evolution of Android malware to better understand emerging attack strategies. Zhou and Jiang analysed thousands of Android malware samples and classified them according to their behavioural characteristics. Their study revealed that Android malware continuously evolves by employing techniques such as privilege escalation, SMS fraud, remote command execution, and information theft. The authors also demonstrated that malware families increasingly adopt code obfuscation and sophisticated attack mechanisms to evade traditional detection techniques, highlighting the need for more comprehensive malware analysis approaches [5].

To support detailed APK inspection, reverse engineering techniques have also been widely adopted in Android malware analysis. Desnos and Gueguen demonstrated how Android applications can be decompiled and analysed using reverse engineering tools to extract application resources, manifest files, Dalvik bytecode, certificates, and source-level information. These techniques enable security analysts to examine application structure, identify suspicious components, and understand application behaviour before installation. However, reverse engineering remains dependent on static application content and cannot directly observe runtime activities performed by malware [6].

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Although existing research has significantly advanced Android malware detection through permission analysis, static analysis, taint analysis, behavioural analysis, and reverse engineering, several limitations still remain. Most existing systems primarily focus on APK analysis and provide limited information regarding applications already installed on users' devices. Furthermore, many frameworks simply classify applications as benign or malicious without presenting comprehensive risk scores, detailed security reports, or practical recommendations that help end users understand the associated security risks.

III. METHODOLOGY

The proposed Android Malware Detection Framework is designed to perform comprehensive security analysis on both Android APK files and applications already installed on a user's device. Unlike conventional malware detection systems that focus only on APK analysis, the proposed system combines static APK analysis with device-level application analysis to provide a complete security assessment. The framework follows a lightweight heuristic-based approach, making it suitable for real-time analysis without requiring root access or high computational resources.

The overall methodology consists of six major phases: data acquisition, application analysis, permission extraction, heuristic-based risk assessment, application classification, and report generation. Initially, users can either upload an Android APK file through the Flask-based web application or scan applications installed on an Android device using the mobile application. The uploaded APK file is processed using APK analysis libraries, while installed applications are analysed using Android PackageManager APIs.

During the analysis phase, the framework extracts important application information including package name, application name, version details, AndroidManifest.xml contents, requested permissions, exported activities, services, broadcast receivers, and content providers. These features provide valuable information regarding the application's behaviour and possible attack surface.

After feature extraction, the framework analyses all requested Android permissions and compares them with a predefined list of security-sensitive permissions. Permissions such as READ_SMS, RECEIVE_SMS, SEND_SMS, READ_PHONE_STATE, READ_CALL_LOG, READ_CONTACTS, and SYSTEM_ALERT_WINDOW are considered high-risk because they are commonly exploited by Android malware to intercept SMS messages, steal personal information, monitor user activities, or perform phishing attacks.

The extracted permissions are then evaluated using a heuristic rule-based risk assessment model. Each sensitive permission is assigned a predefined weight according to its potential security impact. The framework computes the cumulative risk score by summing the weights of all detected sensitive permissions. Based on the calculated score, each application is classified into one of three security categories: Safe, Medium Risk, or High Risk.

Finally, the framework generates a comprehensive security report containing application metadata, detected sensitive permissions, calculated security score, risk level, threat analysis, and security recommendations. This report enables users to understand the security posture of each application and assists them in making informed decisions regarding application installation or removal.

A. *Design Considerations*

The proposed framework is designed based on the following considerations:

- The system should analyse Android APK files before installation.
- The framework should analyse installed Android applications without requiring device rooting.
- Sensitive permissions should be extracted from both uploaded APK files and installed applications.
- Risk assessment should be performed using predefined heuristic rules.
- Applications should be classified into Safe, Medium Risk, and High Risk categories.
- Detailed security reports should be generated to improve user awareness.
- The framework should employ lightweight and open-source technologies suitable for both academic research and practical deployment.

© 2026 IJIRCCE | Volume 14, Issue 7, July 2026 DOI: 10.15680/IJIRCCE.2026.1407057

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

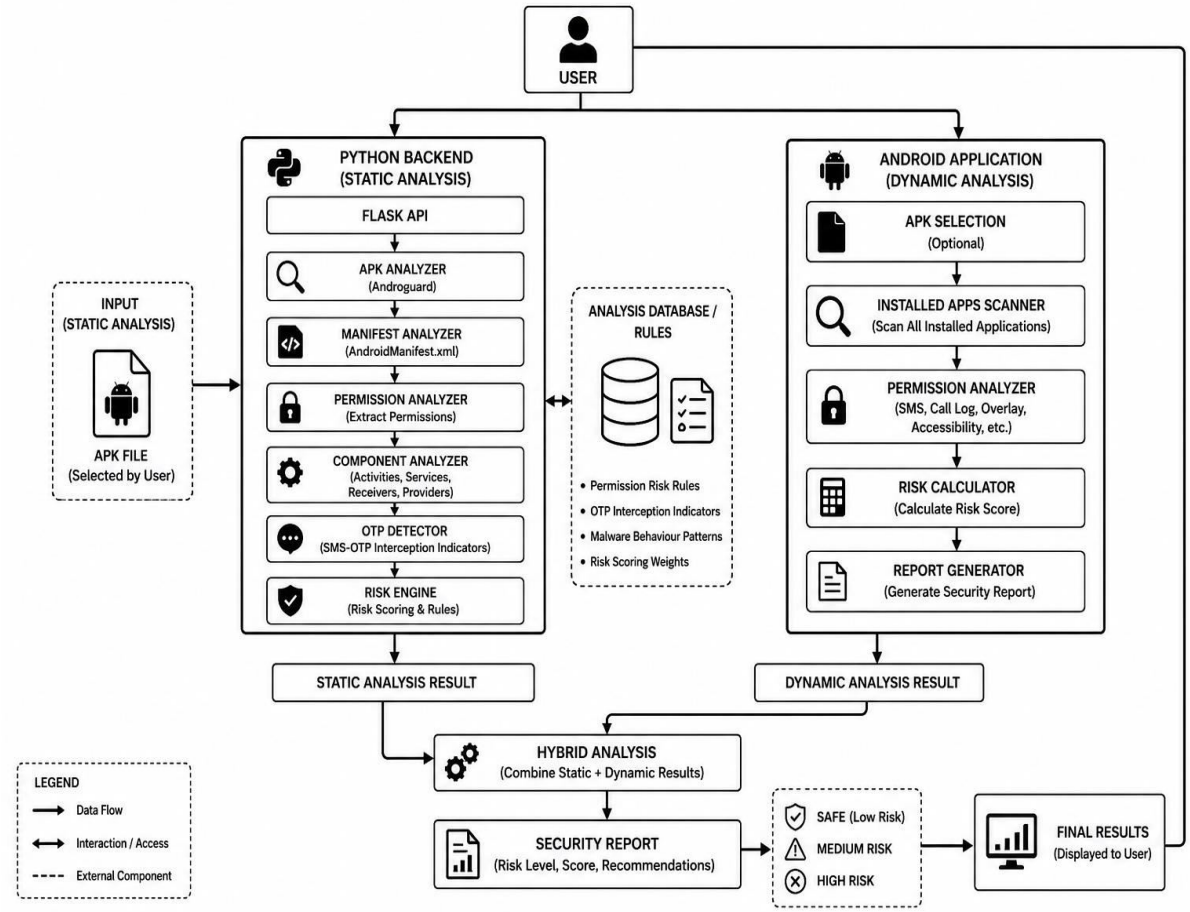


Fig. 1. System Architecture

B. *Description of the Proposed Algorithm*

The proposed algorithm consists of the following sequential steps:

Step 1: Accept an Android APK file through the Flask web application or initiate installed application scanning from the Android application.

Step 2: Extract application metadata including package name, version information, AndroidManifest.xml contents, permissions, activities, services, broadcast receivers, and content providers.

Step 3: Identify security-sensitive permissions such as READ_SMS, RECEIVE_SMS, SEND_SMS, READ_PHONE_STATE, READ_CALL_LOG, READ_CONTACTS, and SYSTEM_ALERT_WINDOW.

Step 4: Compare the extracted permissions with the predefined sensitive permission database.

Step 5: Assign predefined heuristic weights to each detected sensitive permission.

Step 6: Calculate the cumulative application risk score.

Step 7: Classify the application as Safe, Medium Risk, or High Risk according to the calculated score.

Step 8: Generate the final security report containing permission analysis, security score, threat level, and security recommendations.

IV. PSEUDO CODE

Algorithm: Android Malware Detection using Heuristic Permission Analysis

1. Start the application.
2. Accept an APK file from the user or initiate installed application scanning.
3. Extract application metadata.
4. Parse the AndroidManifest.xml file.

© 2026 IJIRCCE | Volume 14, Issue 7, July 2026 DOI: 10.15680/IJIRCCE.2026.1407057

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

5. Retrieve all requested Android permissions.
6. Extract application components including activities, services, broadcast receivers, and content providers.
7. Initialize RiskScore ← 0.
8. Load the predefined list of sensitive Android permissions.
9. Compare extracted permissions with the sensitive permission list.
10. If a sensitive permission is detected, assign its corresponding heuristic weight.
11. Add the assigned weight to RiskScore.
12. Repeat Steps 10–11 for all detected sensitive permissions.
13. Calculate the final application risk score.
14. If RiskScore < T_1, classify the application as Safe.
15. Else if $T_1 \leq$ RiskScore < T_2, classify the application as Medium Risk.
16. Otherwise, classify the application as High Risk.
17. Generate the security score.
18. Generate the threat level.
19. Generate security recommendations based on detected permissions.
20. Display the complete analysis report.
21. Allow the user to save the generated report.
22. Stop the application.

V. RESULTS

The proposed Hybrid Static–Dynamic Analysis Framework for Detecting SMS-OTP Interception Malware on Android Devices was successfully implemented using Python, Flask, Androguard, Java, Android Studio, Android SDK, and Android PackageManager APIs. The developed framework consists of two modules: a Flask-based web application for static APK analysis and an Android application for installed application security assessment.

The static analysis module successfully extracted application metadata, AndroidManifest.xml information, permissions, and application components from uploaded APK files. The extracted permissions were analysed using a heuristic rule-based Risk Engine, which assigned predefined weights to security-sensitive permissions such as READ_SMS, RECEIVE_SMS, SEND_SMS, READ_PHONE_STATE, READ_CALL_LOG, READ_CONTACTS, and SYSTEM_ALERT_WINDOW. Based on the cumulative Risk Score, each application was classified into Safe, Medium Risk, or High Risk categories. The Android application successfully analysed installed applications using Android PackageManager APIs without requiring root access. It retrieved application details, extracted requested permissions, calculated Risk Scores, determined Threat Levels, and generated detailed security reports. The generated reports provided application information, detected sensitive permissions, Security Scores, Risk Levels, and security recommendations, enabling users to understand potential security risks before using an application.

The proposed framework was tested using multiple Android applications with different permission sets. Applications requesting only basic permissions were classified as Safe, whereas applications requesting multiple security-sensitive permissions received higher Risk Scores and were classified as Medium Risk or High Risk. The heuristic-based approach effectively differentiated between low-risk and high-risk applications based on their permission usage. The experimental results demonstrate that the proposed framework provides an efficient, lightweight, and practical solution for Android security assessment. By combining static APK analysis with device-level installed application analysis, the system offers better security visibility than conventional permission-based approaches and helps users identify potentially risky Android applications. Furthermore, the framework generates comprehensive security reports that improve user awareness and support informed decisions regarding application installation and usage.

Application Type	Permissions Analysed	Risk Score	Threat Level	Status
Basic Utility App	Camera, Storage	2	Safe	Low Risk
Social Media App	Contacts, Camera, Storage	5	Medium Risk	Moderate
Messaging App	SMS, Contacts, Phone	10	High Risk	High
Banking App	SMS, Phone, Internet	11	High Risk	High
OTP Security Analyzer	Installed App Analysis	Low	Safe	Successfully Executed

Table 1. Experimental Results

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

VI. CONCLUSION AND FUTURE WORK

This paper presented a Hybrid Static–Dynamic Analysis Framework for Detecting SMS-OTP Interception Malware on Android Devices, which integrates static APK analysis with device-level application analysis to improve Android application security assessment. The proposed framework was successfully implemented using Python, Flask, Androguard, Java, Android Studio, Android SDK, and Android PackageManager APIs. It extracts application metadata, analyses Android permissions, calculates Risk Scores, classifies applications into Safe, Medium Risk, and High Risk categories, and generates comprehensive security reports with appropriate security recommendations. The experimental results demonstrate that the proposed framework effectively identifies applications requesting security-sensitive permissions and provides users with meaningful information regarding potential security threats. The framework is lightweight, user-friendly, does not require root access or specialised hardware, and serves as a practical solution for Android malware assessment and mobile security awareness. Although the proposed framework provides an effective permission-based security assessment, there is scope for further enhancement. Future work includes integrating Machine Learning and Deep Learning algorithms to improve malware detection accuracy, implementing real-time behavioural monitoring for runtime malware detection, analysing network traffic to identify suspicious communications, detecting accessibility-service abuse and overlay attacks, integrating cloud-based threat intelligence databases, and developing an automated alert system for newly installed high-risk applications. Furthermore, the framework can be extended with Explainable Artificial Intelligence (XAI) techniques to improve transparency in risk prediction and support future Android operating system versions. These enhancements will improve the overall efficiency, scalability, and effectiveness of Android malware detection and strengthen mobile device security.

REFERENCES

1. J. Zhao, F. He, Y. Yang, and Y. Zhang, "Identifying Implementation Flaws of SMS OTP Authentication," *IEEE Transactions on Mobile Computing*, vol. XX, no. XX, pp. XXXX–XXXX, 2025.
2. W. Enck, M. Ongtang, and P. McDaniel, "Understanding Android Security," *IEEE Security & Privacy*, vol. 7, no. 1, pp. 50–57, Jan.–Feb. 2009.
3. D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2014.
4. S. Arzt *et al*., "FlowDroid: Precise Context-, Flow-, Field-, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Applications," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2014.
5. W. Enck, M. Ongtang, and P. McDaniel, "Understanding Android Security," *IEEE Security & Privacy*, vol. 7, no. 1, pp. 50–57, Jan.–Feb. 2009.
6. D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2014.
7. S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "FlowDroid: Precise Context-, Flow-, Field-, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Applications," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2014.
8. Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2012.
9. A. Desnos and G. Gueguen, "Android: From Reversing to Decompilation," *Black Hat Abu Dhabi*, 2011.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi crossref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

www.ijircce.com

Scan to save the contact details